Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds

Pedro Silva, Christian Pérez, Frédéric Desprez

Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds

Pedro Silva and Christian Perez
Avalon Team, LIP, INRIA
École Normale Superieure de Lyon
Lyon, France
Email: pedro.silva@inria.fr, christian.perez@inria.fr

Frédéric Desprez
Corse Team, LIG, INRIA
Minatec Campus, 17 rue des Martyrs
Grenoble, France
Email: frederic.desprez@inria.fr

***Abstract*—With the fast growth of the demand for Cloud computing services, the Cloud has become a very popular platform to develop distributed applications. Features that in the past were available only to big corporations, like fast scalability, availability, and reliability, are now accessible to any customer, including individuals and small companies, thanks to Cloud computing. In order to place an application, a designer must choose among VM types, from private and public cloud providers, those that are capable of hosting her application or its parts using as criteria application requirements, VM prices, and VM resources. This procedure becomes more complicated when the objective is to place large component based applications on multiple clouds. In this case, the number of possible configurations explodes making necessary the automation of the placement. In this context, scalability has a central role since the placement problem is a generalization of the NP-Hard multi-dimensional bin packing problem.**

In this paper we propose efficient greedy heuristics based on first fit decreasing and best fit algorithms, which are capable of computing near optimal solutions for very large applications, with the objective of minimizing costs and meeting application performance requirements. Through a meticulous evaluation, we show that the greedy heuristics took a few seconds to calculate near optimal solutions to placements that would require hours or even days when calculated using state of the art solutions, namely exact algorithms or meta-heuristics.

I. Introduction

Cloud computing has become a popular platform for deploying applications as it provides an attractive pay-per-use model and enables any customer to tap into features that in the past were available only to big corporations, including fast scalability, availability, and reliability. We focus on Infrastructure as a Service (IaaS), which consists of providing compute resources, usually as a virtual environment, onto which customers can deploy their applications. The choice of requested resources and even of cloud providers – organizations responsible for providing the infrastructure to users – is up to the application designer, who commonly attempts to balance cost and performance when deploying an application.

The number of cloud providers has grown very quickly to deal with the increasing demand for cloud computing services, and consequently, the number of possible infrastructure configurations to be considered by an application designer has exploded. Hence, placing an application on the cloud – i.e., choosing the best suited ensemble of computers, or more commonly, the best suited set of Virtual Machines (VMs), for an application – becomes a challenge [1].

This challenge is more evident when we consider situations where a large application must be placed in a narrow time constraint. It may arrive that, due to economic advantages, position on marketplace, internal strategy, etc., users are faced with short-term deadlines to execute their large scale applications. In more extreme cases, large crisis management systems, like large scale simulation, data analysis [2] or information management systems [3], may have to be deployed immediately after a disaster. Hence, calculating a placement must not be a time consuming obstacle.

When choosing an environment to host a large application, a designer must choose among possibly thousands of VM types, from different private and public cloud providers, those that are capable of hosting each application part using as criteria application requirements, VM prices, and VM resources. Doing this manually may not be an option, specially if we consider the possibility of deploying multiple application parts on a single VM. Despite the potential economic advantage, this highly increases the complexity of the deployment since the number of possible configurations is exponential. Adding the size of the application and its requirements to the equation leads to an almost intractable problem.

Automating application placement is therefore crucial and has been vastly explored in the literature [1], specially in previous works about cloud brokering [4]. As the placement problem is a generalization of the bin packing problem, which is NP-Hard, meaning that a polynomial-time algorithm to solve it is unknown, scalability becomes a crucial concern.

Most of related work concentrates on solving small to medium sized problems, i.e., problems with a few VM types or small applications. They usually propose solutions based on *exact algorithms*, which do not scale, or *meta-heuristics*, which in spite of being able to give solutions for large problems in feasible time, have their solution quality dependent on the amount of time used to compute it. Works based on more scalable heuristics targeting large problems exist, but despite their important contributions, there are still limitations concerning problem size and cloud model to be treated.

This work addresses the placement of component-based applications on top of environments offered by IaaS. Appli-

cation components or software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system [5]. In general, each component exposes only an interface of communication hiding its implementation and preserving reusability. N-tier services are a clear example of component-based applications where functions are grouped in tiers, or components. Each application tier provides certain functionality to its preceding tier and uses the functionality provided by its successor to carry out its part of the overall request processing [6].

We are interested in the problem of calculating an *initial placement for components of a component based application on multiple clouds* with the objective of minimizing renting costs and meeting performance requirements. Given a set of components describing performance constraints, our objective is to place them in virtual machines rented from possibly many multiple cloud providers meeting the performance constraints and minimizing renting costs. We increase the complexity of placement problems beyond the limits of exact algorithms and meta-heuristics. Hence, we study the use of heuristics, more specifically greedy heuristics, based on first fit decreasing and best fit algorithms, which are scalable and capable of giving near optimal solutions to bin packing problems and its applications.

The remainder of this article is organized as follows. Section II presents the problem characteristics and Section III reviews the literature. Section IV presents the greedy heuristics we used to solve the problem. The evaluation of those heuristics is presented in Section V while conclusions and perspectives are discussed in Section VI.

II. Initial Placement of Component-Based Applications on Multiple Clouds

This work addresses the problem of computing an initial placement for component-based applications on possibly multiple clouds. An instance of the problem comprises a set of application components, or just *components*, for short, that must be placed on *virtual machines* rented from possibly multiple *cloud providers* who offer a set of *virtual machine types*. Each component has *requirements* that must be satisfied by the *capacity* of a rented virtual machine on which it will be placed. To satisfy a placement constraint, a capacity must be larger than a requirement. Examples of requirements or capacities are: 100 MB of RAM, 10 GB of disk storage, 200 Flops of processing, etc. Thus, a problem instance can be summarized as a number d of requirements and capacities, which we will call *dimensions*, a set with n *components* and another set with v *virtual machine types*. The following subsections give more details and formalize the problem.

A. Problem Statement

Let $\mathcal{I}$ be a set of components, $\mathcal{T}$ a set of virtual machine types with D resources (or dimensions) of interest. Let $r_{i,d}$ be the requirements of component i on dimension d, $c_{t,d}$ the capacity of VM type t on dimension d and p_t the price of renting VM of type t per hour. We consider that there is no limit on the number of VM instances for any VM. No component can be assigned to more than one VM, but each VM may hold various components. The objective is to assign all components to VMs so that the requirements of each component is met, the capacities of each VM are respected, and the renting cost is minimized.

We do not consider *network usage* by components and *data locality* in our model. They are important problem parameters, but as component placement and network placement are both NP-Hard problems, the latter is left for future work. As this work considers the initial placement, we do not assume: *à priori* information concerning expected workload, dynamic actors that would allow online modifications of the placement, and renting times. These factors are also left for future work.

B. Optimization Problem Formulation

Let $v_{k,t}$ be the k-th rented VM of type t. Notice that $1 \leq k \leq |\mathcal{I}|$. If we consider that only VMs of type t are rented, then at most $|\mathcal{I}|$ of them will be needed – case where there is only one component per VM. Hence, the set containing all possible rented VMs $\mathcal{V} = \{v_{k,t} \mid 1 \leq k \leq |\mathcal{I}|,\ \forall t \in \mathcal{T}\}$ has size $|\mathcal{V}| = |\mathcal{I}| \times |\mathcal{T}|$.

To simplify the notation, let $v \in \mathcal{V}$. Hence $\exists!\ k$ where $1 \leq k \leq |\mathcal{I}|$ and $\exists!\ t \in \mathcal{T}$ such that $v = v_{k,t}$. Let $c_{v,d}$ be the capacity of dimension d of rented VM v, i.e., $c_{v,d} = c_{t,d}$ and p_v the price paid for renting VM v, i.e., $p_v = p_t$.

Let $m_{i,v} = 1$ if a component i is assigned to a rented VM v, and 0 on the contrary. Let $a_v = 1$ if v were assigned to at least one component and 0 on the contrary.

The optimization problem is described in Equation II-B. Constraint (i) guarantees that each component is assigned to at most one VM, (ii) ensures that no instantiated VM has more components than it can host, (iii) guarantees that $a_v = 1$ when there is at least one component assigned to v.

$$\text{Minimize} \quad \sum_{v \in \mathcal{V}} p_v.a_v$$

s.t.

$$\sum_{v \in \mathcal{V}} m_{i,v} = 1 \qquad \forall i \in \mathcal{I} \qquad (i)$$

$$\sum_{i \in \mathcal{I}} m_{i,v}.r_{i,d} \leq c_{v,d} \qquad \forall v \in \mathcal{V},\ 1 \leq d \leq D \qquad (ii)$$

$$a_v = \begin{cases} 1 & \text{if } \sum_{i \in \mathcal{I}} m_{i,v} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in \mathcal{V} \qquad (iii)$$

$$m_{i,v} \in \{0,1\}$$

$$a_v \in \{0,1\}$$

III. Related Work

This work investigates efficient algorithms to compute an initial placement for very large distributed component based applications on multiple clouds. We consider that we are dealing with thousands of VM types from tens of different cloud providers, tens or hundreds of application components

and tens of performance constraints. Exact algorithms and meta-heuristic approaches would consume long execution times to solve this kind of problem since the placement is a generalization of the bin packing problem, which is NP-Hard.

More specifically, placing component-based applications on the cloud is an instance of the *cost-aware multi-dimensional bin packing problem with heterogeneous bins*, which, is a generalization of traditional multi-dimensional bin packing. In this problem, given a set of n-dimensional items and a group of n-dimensional bins, it is necessary to assign all items to bins using the least number of bins possible. In its cost-aware version, each bin has an *opening price*, and the objective becomes spending the least possible. The mapping between the cost-aware multi-dimensional bin packing problem with heterogeneous bins and the initial placement of component based applications on the cloud is direct. Items are components, bins are VM types, item dimensions are component requirements and bin dimensions are VM capacities. The opening price of a bin is the price of renting a VM.

To the best of our knowledge, no work discusses the exact problem posed here, but since bin packing and more specifically the multi-dimensional bin packing problem and their applications have been vastly explored, there is interesting related work that can be used as starting point to design a solution to our problem. We divided the related work into three groups based on their solution strategies: exact algorithms, meta-heuristics and greedy heuristic.

A. Exact Algorithms Based Strategies

In [7], a solver which uses column generation and branch and bound algorithms to solve multiple type, two dimensional bin packing problems is presented. [8] uses a mixed integer programming (MIP) solver to calculate application placements on VMs, VM resource allocation and consolidation, meeting SLA constraints. In the latter, the number of dimensions is raised to four (CPU, memory, I/O, and bandwidth) in comparison to the former, however only experiences with at most 20 VM types are performed during evaluation. On the same subject, authors in [9] utilize a MIP solver to the problem of VM consolidation aiming at satisfying application SLAs and limiting the number of VM migrations. Also, they allow for a large number of dimensions, approximating their problematic to ours. In [10], a MIP solver is used on a control theory based approach to dynamically calculate the resource allocation for adaptive applications.

Exact algorithms are capable of giving optimal solutions, but when dealing with NP-Hard problems, they all suffer from scalability issues. Depending on the size of the problem, the execution time from an exact algorithm can easily be in the scale of days, as discussed in Section V-B1. Also, except for [8], the cited work is not cost-aware, i.e. none of the solutions considers a price associated to opening a bin. We address this limitation in our approach and use a MIP solver to generate optimal solutions to evaluate the proposed heuristics.

B. Meta-Heuristic Based Strategies

A common approach to address bin packing, and consequently placement related problems, is the usage of meta-heuristic strategies, like genetic algorithms, particle swarm optimization, ant colony optimization and so on. An usual strategy identified in [7], [11], and [12], is the usage of the objective function and constraints of linear programming problems as fitness function / energy function and selection criterion / cooling strategy respectively for genetic algorithms and simulated annealing.

In [13], an approach to do the placement of workflows tasks on the cloud using a genetic algorithm is presented. However, in spite of considering the problem of data locality, it models only two resources and it is implicit that workflow tasks and virtual machine types must be similar. On the same subject, but also addressing the task and virtual machine homogeneity issues, authors in [14] propose a particle swarm optimization based strategy. However, the very high computation complexity of the algorithm is not adequate to our objectives. The same issue characterizes [15], which uses an ant colony optimization approach to calculate workload placement on the cloud.

Despite of commonly finding near-optimal solutions, meta-heuristic based algorithms have their solution quality constrained to the available execution time, meaning that, for large problems, the necessary time to output a near optimal solution may be unfeasible. Also, this type of algorithm usually heavily depends on several application specific tuning parameters to work well. We address huge problem instances that must be solved in feasible time, consequently, this solution is not adequate. Also, when compared to greedy heuristics, the execution time of meta-heuristics are orders of magnitude slower (seconds versus hours) and often the quality of solutions does not follow this proportion, as discussed in [7] and [15].

C. Greedy Heuristics Based Strategies

The usage of greedy heuristics and more specifically best fit and first fit decreasing based approaches are known to be very good options to calculate near optimal solutions to the bin packing problem in feasible time.

In short, the first fit decreasing algorithm sorts items in decreasing order of size and them assigns them to the *first* bin they fit into. Best fit works in a similar manner, but in general, it also sorts bins in increasing order of size aiming at assigning the largest items to the smallest suitable bins.

Additionally, first fit decreasing solutions are proved to be least $\frac{11}{9}$ OPT [16] for one-dimensional bin packing problems. Works like [17] and [18] which present, respectively, a best fit based algorithm for the resource allocation of real-time applications and a first fit based algorithm to deal with placement and elasticity issues, take advantage of this result.

However, to our knowledge, when dealing with heuristics for the multi-dimensional bin packing problem, there is not such a strong evidence of solution quality like a mathematical proof. In spite of that, various works indicates that, in practice, those algorithms are capable of giving very good results.

When dealing with multi-dimensional bins, sorting items or bins becomes a difficult task. To solve this, [19] proposes different procedures of measuring or giving an score to multi-dimensional elements. The authors proposed a function that receives a multi-dimensional input and returns a scalar. We point out that if we consider the size of an item or bin as a *utility*, thus, those functions may be seen as *utility functions*.

[20] proposes a hierarchical resource model and a best-fit based heuristic to map processes onto machines which describe CPU and network requirements and needs. The objective is to minimize communication costs, by assigning communicating processes together inside a same structure hierarchy. Despite considering network bandwidth usage and having up to two dimensions, the algorithms proposed take as an assumption that all machines inside the same hierarchy are homogeneous. This same limitation affects [21], which presents the First Fit Windowed Multi-Capacity, an algorithm for the multi-dimensional bin packing problems that assigns items to bins in order to balance the usage of bin dimensions. [19] presents the Best Fit Dot Product, First Fit Weighted Sum based algorithms and the process, that would be latter named *measure* in [22], of measuring multi-dimensional items or bins. [7] presents the First Fit Ordered Deviation, an algorithm that deals with homogeneity issues but manages only to output solutions to two-dimensional bin packing problems.

[22] and [23] contribute with very interesting ameliorations to the greedy heuristics and measures presented in [19] and also they propose the Priority measure. In spite of that, their algorithms do not consider bin prices.

From the greedy heuristics based on first fit or best fit algorithms and measures discussed in this section, we describe in more details those which presented the most promising results in our tests. Let D be the number of dimensions of the problem, i an item, b a bin and $\mathcal{B}$ the set of bins. i_d and b_d are the values of dimension d of i and b respectively.

- **Measure Weighted Sum [19], [22]:** This measure uses the weighted sum of dimension values, as described in Equation 1.

$$\mathcal{M}_{\text{ws}}(i) = \sum_{d=1}^{D} \alpha_d . i_d,\ 1 \leq d \leq D \tag{1}$$

  α_d is a scaling vector that can assume the following values: 1, $\frac{1}{C_d}$, $\frac{1}{R_d}$, and $\frac{R_d}{C_d}$ where $C_d = \sum_{b \in \mathcal{B}} b_d$ and $R_d = \sum_{i \in \mathcal{I}} i_d$.
- **Measure Priority [22]:** This measure uses the maximal normalized value of dimensions, as described in Equation 2.

$$\mathcal{M}_{\text{p}}(i) = max(\frac{i_d}{\sum_{b \in \mathcal{B}} b_d}),\ 1 \leq d \leq D \tag{2}$$

- **First Fit Windowed Multi-Capacity [21]:** The basic idea of this heuristic is to assign items to bins aiming at balancing bins capacities and items requirements through a ranking matching process.
- **Best Fit Dot Product [19]:** This heuristic uses the dot product between items and bins dimensions as a measure. Items are assigned to bins that maximizes the dot product, as described in Equation 3.

$$\mathcal{M}_{\text{dp}}(i) = \sum_{d=1}^{D} i_d . b_d, \forall b \in \mathcal{B},\ 1 \leq d \leq D \tag{3}$$

D. Discussion

The discussed literature has shown that the cost-aware multi-dimensional bin packing problem with heterogeneous bins and its applications has important open challenges. We discussed an extensive bibliography about the multi-dimensional bin packing, a subproblem of our problem – when bin prices are all equal –, and despite the many contributions from those related works, we identified a range of issues that limit the usage out of the box of their proposed algorithms. Obstacles like cost-obliviousness and homogeneity of bins or items are the main limitations that we had to overcome to target our problem. In Section IV, we detail the chosen greedy heuristics and the necessary changes we implemented to solve our problem. In Section V, we evaluate their performance in terms of solution quality and execution time.

IV. Improved Greedy Heuristics

In Section III-C, we discussed a set of greedy heuristics created for solving the multi-dimensional bin packing problem (MDBPP). Due to limitations, however, it is not possible to use MDBPP greedy heuristics directly for solving the cost-aware MDBPP with heterogeneous bins.

This section discusses the changes we made that enabled us to use MDBPP greedy heuristics to address our problem. In summary, we modified existing algorithms so that they consider *opening prices* and *heterogeneity* of bins.

A. Cost-Awareness

In the traditional MDBPP, the cost of a solution is simply measured in terms of number of open bins. This is equivalent to consider that all bins are free. However, in the cost-aware MDBPP with heterogeneous bins, there is a price for opening bins and they may vary, thus, MDBPP greedy heuristics must be adapted to allow for bins with heterogeneous prices.

This means that, to solve the cost-aware MDBPP, it should try to assign items to the *most profitable* bins first. For example, in the cost-aware MDBPP a solution with ten \$1 bins of size s is better than a solution with one \$20 bin of size $10s$. Both solutions manage to assign all items to bins, but the former costs \$10 and the latter \$20. We can observe that the ratio $\frac{capacity}{price}$ of the first solution's bin is larger than that of the second solution's bin. Hence, sorting the list of bins by the *ratio between price and capacity* is essential.

To accomplish this task we use the concept of measures (see Section III-C) where a *measure* is a function that receives as input a multi-dimensional vector (the representation of a bin or item dimensions) and outputs a *score* or the *size* of that vector. Measures are originally used for sorting items in decreasing

order, but we use it to sort bins in a way that more profitable bins comes first. Thus, when assigning items to vectors, the first bins to be scanned would be the most profitable.

It is also important to open bins the least often possible, hence, using the capacities of already open bins is imperative. To do so, before looking for new bins, we verify if items can be assigned to already open bins, by scanning them, in decreasing order of size.

Equations 4 and 5 describes the weighted sum, which is the measure we use to sort bins and open bins.

$$\mathcal{M}_{ws}^{\text{bin}}(t) = \frac{1}{p_t} \sum_{d=1}^{D} \alpha_d . c_{t,d} \tag{4}$$

$$\mathcal{M}_{ws}^{\text{open bin}}(v) = \frac{1}{p_v} \sum_{d=1}^{D} \alpha_d . \sum_{i \in \mathcal{I}} m_{i,v} . r_{i,d} \tag{5}$$

We use the coefficient $\alpha_d = \frac{R_d}{C_d} = \frac{\sum_{i \in \mathcal{I}} r_{i,d}}{\sum_{t \in \mathcal{T}} c_{t,d}}$ as a scaling vector. We do not detail the process of choosing the best coefficients and measures due to space constraints, but essentially it is based on the evaluation of the performance of the greedy heuristics using data gathered from experimenting different combinations of coefficients. This adaptation allowed us to use First Fit Decreasing Priority (FFD-P) on cost-aware MDBPPs.

To make Best Fit Dot Product (BF-DP) cost-aware, we used a similar approach. In this heuristic, the objective is to assign each item to its "most adapted" bin, which, in this case corresponds to the bin that maximizes the dot product between item and bins. To make the algorithm cost-aware, it was necessary to consider bin prices when calculating the dot product. We use a strategy to prioritize the usage of open bins similar to the one used for FFD-P. A dot product between unassigned items and open bins is calculated before opening a bin. Those adaptations are described in Equation 6 and 7 .

$$\mathcal{M}_{\text{dp}}^{bin}(i) = \frac{1}{p_t} \sum_{d=1}^{D} r_{i,d} . c_{t,d}, \ \forall t \in \mathcal{T} \tag{6}$$

$$\mathcal{M}_{\text{dp}}^{open\ bin}(i) = \frac{1}{p_v} \sum_{d=1}^{D} r_{i,d} \sum_{i \in \mathcal{I}} m_{i,v} . r_{i,d}, \ \forall v \in \mathcal{V} \tag{7}$$

B. Heterogeneous Bins

Among the heuristics in which we are interested, First Fit Decreasing Windowed Multi-Capacity (FFD-WMC), considers that bins are homogeneous.

FFD-WMC assigns items to bins with the objective of balancing the usage of dimensions through a rank matching mechanism (see Section III-C). Ranks are calculated using an empty bin as basis of comparison and they express the percentage of used dimensions. It is essential that ranks share the same base so they can be compared. Our strategy to allow for heterogeneous bins, is to construct a maximal bin, which is composed by the largest dimension capacities from all bins and use it as basis of comparison. Thus, ranks dimensions become percentages of these maximal bin dimensions.

Clearly, now that bins may be heterogeneous it is also necessary to introduce price heterogeneity to the problem. To do so, we make use of the approach presented on Section IV-A and sort the bins by decreasing $\frac{capacity}{price}$ ratio, using the weighted sum (see Section 1) as a measure. Thus, the more profitable bins would be scanned first.

	d	n	v
A	$1, \ldots, 8$	$1, \ldots, 19$	$100, 200, \ldots, 1000$
B	$1, \ldots, 8$	$10, 20, \ldots, 100$	$1000, 1100, \ldots, 10000$

TABLE I
EXPERIENCE CLASSES.

V. EVALUATION

This section evaluates the performance of greedy heuristics for calculating an initial placement for component based applications on multiple clouds. To achieve this, we compare these greedy heuristics to two state of the art solutions, namely, a MIP solver and a simulated annealing meta-heuristic, in different scenarios, and analyze the results. Before going into that, it is important to present how the necessary experiments were performed and how their input data were generated.

A. Methodology

An *experiment* is the resolution of a set of placement *problem instances* by a set of algorithms within a given *timeout*. A problem instance, as discussed in more details in Section II, is composed by a group of n components and a group of v virtual machine types, both describing d dimensions requirements and capacities, respectively.

There are two classes of experiments, *A* and *B*, distinguished by problem instances sizes as described on Table I. The small to medium problem instances from Class A will be used to evaluate the performance of greedy and meta heuristics against the exact algorithm, as the latter is not scalable. Class B is composed by large problem instances which will be used to evaluate the greedy heuristics against meta-heuristics.

To construct the problem instances it is necessary to generate the values of VM capacities, prices and component requirements. The procedure we use is the generation of pseudo-random values picked uniformly inside an interval using the method *randint* from python's module *random*. Table II presents in detail those intervals.

Dimension	**Requirements**	**Capacities**
(i)	800 $\sim$ 3000	1000 $\sim$ 3500
(ii)	1 $\sim$ 16	2 $\sim$ 32
(iii)	1 $\sim$ 32	2 $\sim$ 40
(iv)	50 $\sim$ 3500	150 $\sim$ 4000
(v)	5 $\sim$ 30	10 $\sim$ 80
(vi)	1 $\sim$ 8	1 $\sim$ 16
(vii)	1 $\sim$ 10	5 $\sim$ 40
(viii)	10 $\sim$ 80	10 $\sim$ 80

TABLE II
INTERVALS OF DATA GENERATION.

To generate the VM renting prices, we use the capacities from the first 4 VM type dimensions, in a way that, the larger they are, the more expensive is the renting price. We simulate different prices from different cloud providers through the generation of pseudo-random coefficients – as before, using the method *randint* – from predefined intervals. The price of a VM type $p_t = \alpha+\beta+\gamma+\delta$ where $\alpha = c_{i,1} \times randint(1,3)$; $\beta = c_{i,2} \times randint(8,20)$; $\gamma = c_{i,3} \times randint(5,8)$; $\delta = c_{i,4} \times randint(10,15)$, if $c_{i,4} \leq 500$, otherwise $\delta = c_{i,4} \times randint(20,25)$.

The three greedy heuristics we are evaluating are the following: Best Fit Dot Product, First Fit Decreasing Priority, and First Fit Decreasing Windowed Multi-Capacity. Those algorithms were introduced in Sections III and IV. Our test platform and greedy heuristics were developed in Python.

Experiments were conducted on Dell PowerEdge R720 (2 CPUs, 6 cores) and AMD Opteron 6164 HE 1.7GHz (2 CPUs, 12 cores) from *Taurus* and *Stremi* clusters from *Grid'5000* [24].

B. MIP Solver and Simulated Annealing Analysis

We are interested in evaluating the performance of our greedy heuristics with the very large Class B (see Table I) problem instances. It would be interesting to compare the solutions from greedy heuristics to the optimal of each problem instance, however, as we are dealing with a NP-Hard problem, this is unfeasible.

Our strategy, then, is to calculate the optimal solution of each problem instance from a Class A experiment (see Table I) giving the MIP solver 30 hours per instance to do this task. Using these data, we validate the performance of a scalable meta-heuristic, which in our case is simulated annealing, and use it as a baseline algorithm to analyze the performance of the greedy heuristics on the large Class B problem instances.

1) MIP Solver: To evaluate the performance of MIP solvers, we integrated to our test platform the *SCIP* solver [25], a framework for constraint integer programming and branch-cut-and-price (the formulation is described in Section II-B).

We conducted one Class A experiment using SCIP with a timeout of 30 hours. The framework managed to calculate the optimal for around 34% of all Class A problem instances. These solved problem instances are mainly characterized by having a small number of application components, virtual machine types and dimensions. This performance is expected as we are dealing with a NP-Hard problem. Even if we consider that there is room for improvement of the solver performance by optimizing the modeling or by using a faster solver, we would still expect a low rate of solved instances due to the nature of the problem we are dealing with. Hence, despite giving optimal solutions, using a MIP solver is not an option when resolution time is a strong constraint.

2) Simulated Annealing: As discussed in Section III, using meta-heuristics to solve problems similar to the bin packing problem is a very common approach. Among all algorithms of this type, like genetic algorithms, particle swarm optimization, ant colony optimization and others, we choose the simulated annealing because, in addition to successful experiences in other similar contexts [7], it has less configuration parameters, hence it is easier to apply it to our problem.

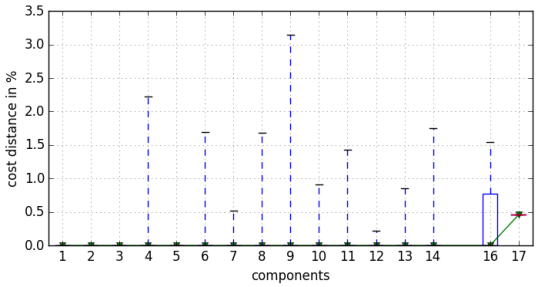

Fig. 1. Distances between solution costs from MIP solver and simulated annealing aggregated by number of components for Class A experiments.

We used the *Simanneal* [26] module, which is written in Python and was easily integrated to our test platform. Also, we conducted one Class A experiment and 2 sets of Class B experiments always using a timeout of 10 minutes per problem resolution. We use 10 minutes instead of the 30 hours given to SCIP solver because 10 minutes is more realistic and also because during our tests we noticed that in most of cases, after this time, the solution improvements became scarcer. The algorithm parameters were optimized for computing a solution within that timeout by the auto-tuning tool included in *Simanneal* module.

We observed that simulated annealing managed to output a solution for all problem instances in less than ten minutes. From the around 34% of problem instances whose optimal solution was known, the simulated annealing algorithm managed to achieve it on around 97% of the cases, as illustrated in Figure 1. To construct this graph, we grouped all Class A problem instances by number of components and plotted, in form of box-plot, the distances (or differences) between solution costs from MIP solver and simulated annealing. Throughout this work we will always group solutions by number of components because this parameter showed to be the one that most interferes on solution cost in a consistent way. We can notice that in spite of the small variation observed as the number of components grows, the distance is always smaller than 3% and the median is always zero, except when the number of components is 17.

Even if it was only possible to compare the solutions from simulate annealing to the optimal in a reduced portion of the experiments, we have a promising indication of the capabilities of this meta-heuristic. This justifies the usage of simulated annealing as baseline in our further analysis and also the success of that meta-heuristics to solve this kind of problem.

C. Greedy Heuristics

In this section, our objective is to evaluate the performance of our greedy heuristics. The main goal is evaluating them using the large Class B experiences, however, we also investigate the performance of our greedy heuristics using the 34%

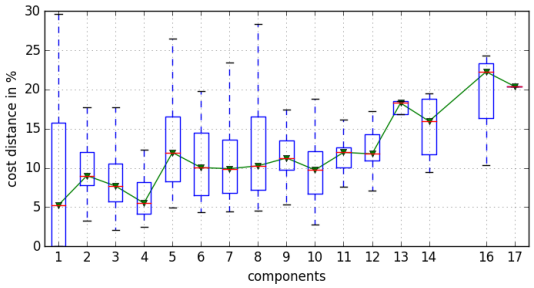

Fig. 2. Distances between optimal solution costs and greedy heuristic group's aggregated by number of components for Class A experiments.

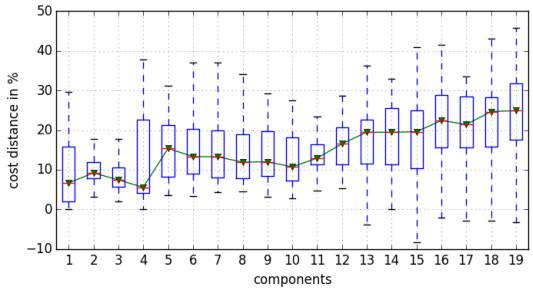

Fig. 3. Distances between solution costs from simulated annealing and greedy heuristic group aggregated by number of components with Class A experiments.

of Class A experiments whose optimal solution is known. The considered greedy heuristics are Best Fit Dot Product, First Fit Decreasing Priority, and First Fit Decreasing Windowed Multi Capacity. For more details about them, see Section IV.

Our evaluation strategy is, at first, analyzing the performance of the greedy heuristics *together*, i.e., comparing them in group to other algorithms and then, in a second moment, evaluating them individually. We consider that the greedy heuristics are executed sequentially, thus, when *group comparing*, the execution time is the sum of the execution times from all involved greedy heuristics. Then, we keep only the best cost – less expensive – among all greedy heuristics costs.

Throughout this section we use data gathered from one Class A and two Class B experiments with a timeout of 10 minutes to solve each problem instance. Class A and Class B experiments are used to evaluate greedy heuristics against the MIP solver and simulated annealing respectively. To analyze the two Class B experiments results, we take the solutions from each problem instance in each of the experiments and calculate the average of costs and execution times.

1) Group analysis – Class A Experiment: At first we compare solutions from the group of heuristics to the available 34% of optimal solutions from Class A problem instances. Figure 2 illustrates the *cost distance* between the group of greedy heuristics solutions and optimal values. The distances are aggregated by the number of components from the solved problems and the median is represented by a solid curve.

The first thing to notice is that despite giving very few optimal solutions (about 3.4%), the greedy heuristics group managed to output solutions at most 30% more expensive than the optimal with a median varying from 5.52% to 22.25%. These measures are consistent with those found when comparing the greedy heuristics group to simulated annealing, also using Class A experiments, as illustrated in Figure 3. In this case, the median varies between 5.56% and 24.88% and distances between -8.22% and 45.86%. It is important to highlight that in Figure 3 we plot 100% of Class A problem instances since simulated annealing managed to give solutions to all feasible problem instances.

Negative distances (around 3.18%) refer to situations where greedy heuristics group managed to output a better solution than simulated annealing. Finally, in both graphs it is possible to identify a degradation of greedy heuristics group solutions as the number of components raises, specially when the number of components is greater than 12.

To complete this first analysis, it is important to compare the execution times from the algorithms. Nevertheless, due to space constraints, we let the evaluation of the execution time from simulated annealing to Section V-C2.

When comparing the execution times from the solver and greedy heuristics, we observe a huge difference: the solver took around 3 hours and 36 minutes to solve about 34% of Class A problems while the greedy heuristics group consumed 23.21 seconds to give solutions to all Class A problems. It is important to remember that the solver was given a timeout of 30 hours per problem instance to solve Class A problems.

There are some preliminary conclusions taken from this first analysis. The most important is that, the quality of solutions is at most 30% worst than the optimal but they are calculated at least 10 times faster, indicating a reasonable solution quality. Also, being at most 30% worst than optimal is only around 8% greater than the 11/9 OPT [16] (see Section III-C) or 22.22% of the optimal proved to be the ceil of first fit decreasing solutions for the one-dimensional bin packing problem. Finally, when comparing Figure 2 with Figure 3 it is possible to identify that both distance medians follow a similar pattern, which indicates that simulate annealing would be an interesting choice as a baseline algorithm.

2) Group analysis – Class B Experiments: Figure 4 illustrates the cost distance between the group of greedy heuristics and simulated annealing. Solutions are aggregated by the number of components from the solved problem instances and the evolution of the median is represented by a solid curve. It is possible to observe that the greedy heuristic group managed to output a better solution than simulated annealing to around 15% of problem instances. One can notice that it happens more frequently when the number of components is bigger than 70. This is observed because the timeout of 10 minutes is not sufficient for simulated annealing to calculate a better solution as the size of problem instances grows.

In the remainder 85% of problem instances, where simulated annealing outputs better solutions, we can also observe that although the distances are always smaller than 40%, the

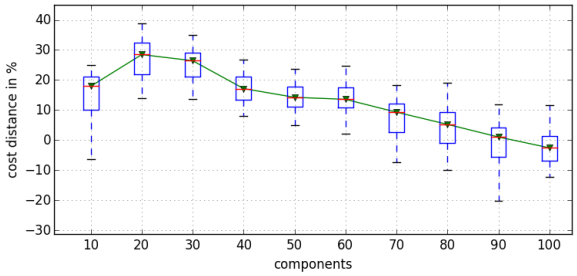

Fig. 4. Distance between simulated annealing solution costs and greedy heuristic group aggregated by number of components.

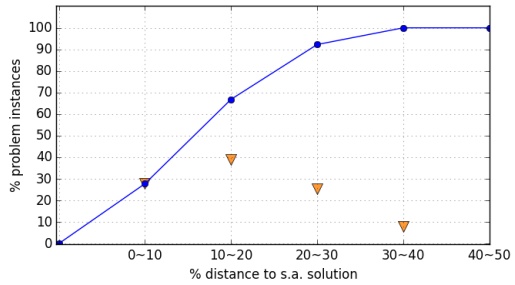

Fig. 5. Distance between solution costs from simulated annealing solutions and greedy heuristics group on problem instances where simulated annealing solutions were better.

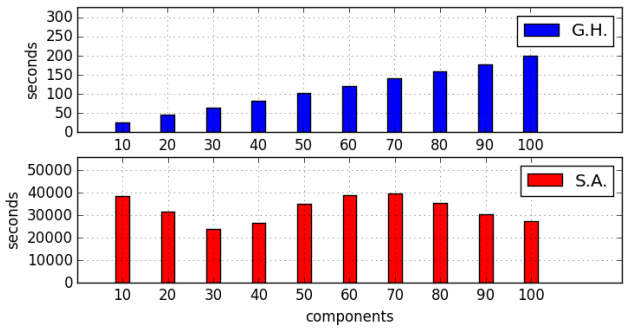

Fig. 6. Sum of execution times from greedy heuristics group (above) and simulated annealing (bellow) to solve all problem instances. Results are aggregated by number of components.

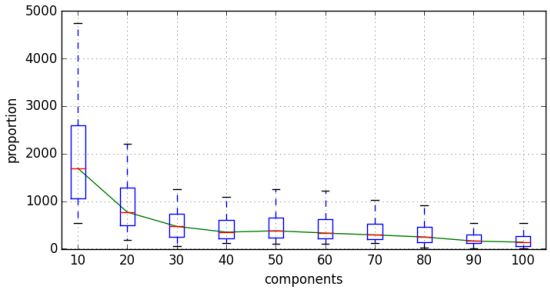

Fig. 7. Ratio between simulated annealing and greedy heuristics execution times aggregated by number of components.

median never exceeds 30%.

Figure 5 helps us to have a better understanding on how worse was the solution cost of greedy heuristics on these 85% of problem instances where simulated annealing calculated better solutions. The Y-axis is the percentage of solved problem instances and the X-axis is the cost distance between the greedy heuristic group and simulated annealing. The solid curve is the aggregated percentage of problem instances. We can observe that around 40% of solutions are between 11% and 20% worse than simulated annealing and, most importantly, that around 95% of solutions are at most 30% worse than simulated annealing's ones. This clearly indicates that, depending on the application requirements, the degradation of solution quality may not be very significant, especially when taking into account the difference between execution times from the group of greedy heuristics and the simulated annealing which will be discussed in the following lines.

In Figure 6 the execution times to solve problems with the same number of components are summed up. While the sum of execution times from greedy heuristics vary from 25 to 210 seconds, simulated annealing's ranges from 26200 to 42185 seconds or from 7.3 to 11 hours.

This can be better seen in Figure 7, which aggregates the ratio between greedy heuristics group and simulated annealing execution times by number of components and plot this data as a box-plot. We are using ratios instead of distances percentages because of the huge gap between values. We can see that the sum of greedy heuristic group execution times is at least 10 times and at most around 4750 times faster than simulated annealing's ones. We can also observe that the median is always greater than 138 and as the number of components grows, the efficiency of the greedy heuristics group reduces.

It is well known that the usage of heuristics involves a trade-off between solution quality and execution time. The analysis of the performance of our greedy heuristics as a block indicated that even with an execution time between 10 to 4750 times smaller, the greedy heuristics managed to output good quality solutions and sometimes better solutions than simulated annealing's for large problems.

3) Individual Analysis: In this section, we evaluate the greedy heuristics individually using an average of two Class B experiments. Our objective is to understand their behavior and also to investigate how the reduction of the group of greedy heuristics would affect the quality of solutions.

Figure 8 illustrates the percentage of best, second best, and third best solutions per greedy heuristic. We notice that First Fit Decreasing Priority (FFD-P), First Fit Decreasing Windowed Multi-Capacity (FFD-WMC), and Best Fit Dot Product (BF-DP) have the best solutions for around 56%, 29% and 30% of problem instances respectively. Even if BF-DP has a relatively small number of best solutions, it manages to output second best solutions to almost 57% of problems. Thus, BF-DP gives the best or second best answer to around 74% of all problems. FFD-P and FFD-WMC manage to do

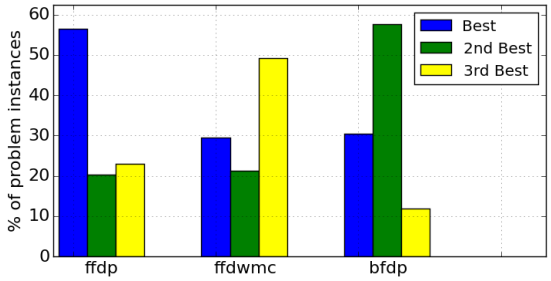

Fig. 8. Percentage of best, second best and third best solutions by greedy heuristic.

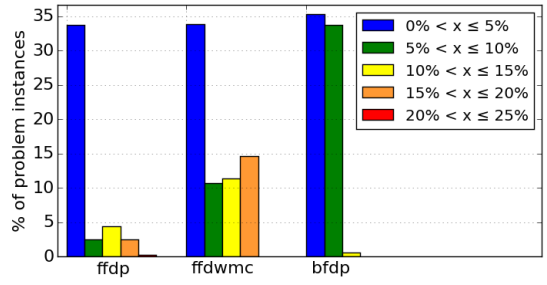

Fig. 9. Distance in % from best greedy heuristic solution per algorithm and number of occurrences in % of problem instances.

the same to 76% and 50% of problem instances, respectively.

Figure 9, which illustrates the percentage of problems where a greedy heuristic did not have the best solution and the distance to it, helps us understand the quality of non best solutions. The most important aspect to notice in this graph is that, for all considered algorithms, the maximum distance to the best solution is bellow 25% and that, in most of the cases, it is bellow 10%. Also, one can notice that BF-DP manages to give a solution at most 10% worst than the best greedy solution for 99.20% of problem instances where it does not output the best solution. Concerning the 44% of problem instances where FFD-P did not give the best solution, it was capable of giving solutions 5%, 10%, 15%, 20% and 25% worst than the best solution to 33.75%, 2.5%, 4.4%, 2.5% and 0.27% of problem instances respectively. FFD-WMC, by its turn, managed to output solutions 5%, 10%, 15%, 20% worst than the best solution to 33.8%, 10.7%, 11.38% and 14.58% of problem instances, respectively.

This first analysis based only on solution cost quality indicates the superiority of FFD-P and BF-DP against FFD-WMC, however, to have a better understanding, it is necessary to verify their individual execution times too.

Table III summarizes the individual execution times from the considered greedy heuristics. Those values are the sum of the execution times used to compute solutions to all problem instances. We can observe that despite giving good solutions, BF-DP responds for around 90.78% of the sum of greedy heuristics execution times, followed by FFD-WMC and FFD-P, responsible for around 6% and 2.44% respectively. This indicates that it may be interesting to reduce the size of the greedy heuristics group to have a smaller execution time. However, it is also important to verify the impact of this reduction on the solution quality.

Algorithm	**Time (s)**	**Participation**
B.F. Dot Product	1012.535	90.78%
F.F.D. Priority	27.25	2.44%
F.F.D. Windowed Multi-Capacity	75.57	6%

TABLE III
EXECUTION TIMES FROM GREEDY HEURISTICS

Table IV presents a series of metrics related to comparisons between possible combinations of greedy heuristics groups and the original group. It is possible to verify that using only FFD-P improves the execution time in 97.55%. However, doing so also degrades 43.47% of solutions in about 17.38%. Also, we can verify that it is possible to improve the execution time in 90% with a smaller impact over solution quality when using FFD-P and FFD-WMC together. In this case, we observe that around 19.02% of solution costs would suffer a degradation between 0.87% and 4%, in average. Thus, clearly, if it is necessary to improve execution time, the best option is to remove BF-DP from the group of heuristics.

	MAX	AVG	MIN	MED	DEG	IMP
BF	9.93	4.46	0.02	4.64	41.08	9.21
FP	17.38	3.73	0.01	1.97	43.47	97.55
FW	16.34	7.07	0.01	5.56	69.58	93.22
BF & FP	6.2	2.07	0.02	1.38	18.61	6.77
BF & FW	9.93	4.72	0.02	4.73	50.13	2.44
FP & FW	4.00	0.87	0.01	0.58	19.02	90.78
BF & FP & FW	0	0	0	0	0	0

TABLE IV
BF, FP AND FW ARE BF-DP, FFD-P AND FFD-WMC, RESPECTIVELY. MAX, AVG, MIN AND MED ARE MAXIMUM, AVERAGE, MINIMUM AND MEDIAN OF SOLUTION COSTS DISTANCES, IN PERCENTAGE OF SOLUTIONS FROM THE ORIGINAL GROUP OF HEURISTICS. DEG IS THE PERCENTAGE OF SOLUTIONS THAT WERE DEGRADED AND IMP IS THE EXECUTION TIME IMPROVEMENT IN PERCENTAGE OF ORIGINAL EXECUTION TIME.

Finally, we identify 3 possible configurations for the greedy heuristics group: (i) the fastest one, which would be composed uniquely by FFD-P, with an improvement of 97.55% of execution time but with around 43.47% of its solutions degraded, (ii) the medium term, which would be composed by FFD-P and FFD-WMC, with an improvement of 90.78% of execution time but having 19.02% of solutions degraded and, (iii) the slowest configuration, composed by FFD-P, BF-DP, and FFD-WMC which give the best solutions. It is important to notice, however, that "slowest" here means a configuration capable of solving 720 large problem instances in less than 19 minutes. If we go further and analyze execution times per problem instance from Class B, we will find averages of 1.4 s, 0.03 s and 0.1 s for BF-DP, FFD-P and FFD-WMC, respectively, and, at the same conditions, maximum execution times of 5.7 s, 0.23 s and 0.35 s for BF-DP, FFD-P and FFD-WMC, respectively. Thus, extremely short execution times.

VI. Conclusion and Future Work

Cloud computing has changed the way applications are developed and ported over distributed infrastructures. New applications built upon several components have to be deployed over multiple clouds to benefit from many different VMs and offering different renting costs. This is indeed a complicated problem, especially as the complexity of these applications and the number of parameters and features grows.

The main objective of this work was thus to develop fast algorithms to solve the problem of calculating an initial placement for large-scale component based applications over multiple clouds. In addition, we considered that the parameters of this problem (number of VM types, multiple cloud providers, number of components, number of dimensions and objective functions) could be huge, which might prohibit the usage of solutions such as MIP solvers and meta-heuristics.

To achieve that objective we adapted to our problem very efficient greedy heuristics originally conceived to solve the multi-dimensional bin packing problem. After a detailed evaluation, we indicated that our greedy heuristics were capable of giving solutions compatible with meta-heuristics solutions but calculated at least 100 times faster. Certainly, our approach is better suited for situations where there is space for a light degradation of solution quality in exchange of a reduced execution time. It is also possible to use our greedy heuristics solutions as a first solution input for meta-heuristics or exact algorithms. Finally, virtually any application of the cost-aware multi-dimensional bin packing problem with heterogeneous bins may take advantage of our results and algorithms.

We have started incorporating these greedy heuristics in the PaaSage open source integrated platform developped within the European project Paasage[1] to calculate placements for large-scale N-tier applications on the multi-cloud.

In the future, we plan to increment our heuristics to allow taking network bandwidth and data locality into consideration when calculating a placement. Moreover, dynamicity and elastic resource management have to be taken into account. We plan to design semi-static algorithms adapting the resource allocation after a static allocation.

Finally, we see the ascension of edge computing related applications [27] as a source of interesting use cases for our work. Although the problem of placing an edge computing application usually presents less cloud providers and consequently a reduced set of virtual machines, the potential size of applications and their level of distribution are challenging.

Acknowledgment

All experiments were carried out using the Grid'5000 testbed, supported by a group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This work was partially supported by the PaaSage (FP7-317715) EU project.

[1]C.f., http://www.paasage.eu/

References

[1] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, 2014.

[2] S. Voigt, T. Kemper, T. Riedlinger, R. Kiefl, K. Scholte, and H. Mehl, "Satellite image analysis for disaster and crisis-management support," *Transactions on Geoscience and Remote Sensing*, 2007.

[3] M. Kuhnert, O. Paterour, A. Georgiev, K. Petersen, M. Bscher, J. Pottebaum, and C. Wietfeld, "Next generation, secure cloud-based pan-european information system for enhanced disaster awareness," in *ISCRAM*, 2015.

[4] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, 2014.

[5] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2002.

[6] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *SIGMETRICS*, 2005.

[7] B. Han, G. Diehr, and J. Cook, "Multiple-Type, Two-Dimensional Bin Packing Problems: Applications and Algorithms," *Annals of Operations Research*, 1994.

[8] C. Seçinti and T. Ovatman, "On Optimizing Resource Allocation and Application Placement Costs in Cloud Systems," in *CLOSER*, 2014.

[9] H. N. Van, F. Tran, and J.-M. Menaud, "SLA-Aware Virtual Resource Management for Cloud Infrastructures," in *CIT*, 2009.

[10] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," in *HPDC*, 2010.

[11] C. C. T. Mark, D. Niyato, and T. Chen-Khong, "Evolutionary Optimal Virtual Machine Placement and Demand Forecaster for Cloud Computing," *IEEE AINA*, 2011.

[12] D. P. Chandu, "A Parallel Genetic Algorithm for Three Dimensional Bin Packing with Heterogeneous Bins," *International Journal of Computer Trends and Technology*, 2014.

[13] H. Goudarzi and M. Pedram, "Multi-dimensional SLA-Based Resource Allocation for Multi-tier Cloud Computing Systems," in *CLOUD*, 2011.

[14] M. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, 2014.

[15] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," in *GRID*, 2011.

[16] M. Yue, "A Simple Proof of the Inequality $FFD(L) \leq \frac{11}{9}OPT(L)+1, \forall L$ for the FFD Bin-Packing Algorithm," *Acta Mathematicae Applicatae Sinica*, 1991.

[17] K. Kumar, J. Feng, Y. Nimmagadda, and Y.-H. Lu, "Resource Allocation for Real-Time Tasks Using Cloud Computing," *ICCCN*, 2011.

[18] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "Kingfisher: Cost-Aware Elasticity in the Cloud," *INFOCOM*, 2011.

[19] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for Vector Bin Packing," Microsoft Research, Tech. Rep., 2011.

[20] F. Tessier, G. Mercier, and E. Jeannot, "Process Placement in Multicore Clusters:Algorithmic Issues and Practical Techniques," *IEEE Transactions on Parallel and Distributed Systems*.

[21] W. Leinberger, G. Karypis, and V. Kumar, "Multi-Capacity Bin Packing Algorithms with Applications to Job Scheduling Under Multiple Constraints," in *ICPP*, 1999.

[22] M. Gabay and S. Zaourar, "Variable Size Vector Bin Packing Heuristics - Application to the Machine Reassignment Problem," INRIA, Tech. Rep., 2013.

[23] ——, "Vector Bin Packing with Heterogeneous Bins: Application to the Machine Reassignment Problem," *Annals of Operations Research*, 2015.

[24] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Perez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *CLOSER*.

[25] T. Achterberg, "SCIP: Solving Constraint Integer Programs," *Mathematical Programming Computation*, 2009.

[26] M. Perry, "Simanneal: Python module for simulated annealing optimization." [Online]. Available: https://github.com/perrygeo/simanneal

[27] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *AFIN*, 2014.